Available online at www.sciencedirect.com

SCIENCE DIRECT

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 133 (2005) 175–191

www.elsevier.com/locate/entcs

Obtaining Memory-Efficient Solutions to Boolean Equation Systems

Misa Keinänen[1,2]

Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
Helsinki University of Technology
P.O. Box 5400, FIN-02015 HUT, Finland

Abstract

This paper is concerned with memory-efficient solution techniques for Boolean fixed-point equations. We show how certain structures of fixed-point equation systems, often encountered in solving verification problems, can be exploited in order to substantially improve the performance of fixed-point computations. Also, we investigate the space complexity of the problem of solving Boolean equation systems, showing a NL-hardness result. A prototype of the proposed technique has been implemented and experimental results on a series of protocol verification benchmarks are reported.

Keywords: algorithm, boolean equation systems, memory efficient resolution, space complexity.

1 Introduction

Many verification tools and algorithms for finite-state concurrent systems need to solve fixed-point equations, such as those for checking behavioural equivalences and model checking μ-calculus [17]. However, solving general fixed-point equations is a difficult task for which no polynomial time algorithms are known (although the problem is in $NP \cap co-NP$ [11], and even in $UP \cap co-UP$ [16]).

[1] Petteri Kaski is thanked for valuable discussions. Jaco van de Pol provided useful comments on the specifications from [8,13]. The work was financially supported by Academy of Finland (project 53695), Emil Aaltonen foundation and Helsinki Graduate School in Computer Science and Engineering.

[2] Email: mkk@tcs.hut.fi

doi:10.1016/j.entcs.2004.08.064

Special techniques have been developed to overcome this and there are many practically relevant classes of fixed-point equation systems which do have very efficient algorithms [1,4,6,11,12,14,19]. But a remaining problem is the memory consumption of these techniques. Industrial scale verification problems lead typically to systems with millions of fixed-point equations, and these equations may become too large to fit the memory at the same time. Yet, so far no-one has investigated the space complexity of solving fixpoint equation systems, and very little attention has been paid to develop memory-efficient solution algorithms.

In this paper, we investigate memory-efficient techniques for solving Boolean equation systems [1,2,18,20,21,23,26]. Boolean equation systems give a useful framework to study fixed-point computations, because μ-calculus can be encoded in this setting (for example, see [1,5,21,23]). We will call attention to a fragment of Boolean equation systems, which we call *stratified* systems of equations. These are specific fixed-point equation systems where, the variables are interspersed in the equations so that the computation of the solutions becomes very simple, and can be optimized to be memory efficient. We present efficient techniques for solving such systems. The main novelty is that we exploit the particular structure of fixed-point equations to substantially improve the memory usage.

We also study the space complexity of solving general Boolean equation systems and prove a NL-hardness result which appears to be new. This helps to understand the theoretical minimum of space storage needed for solving Boolean equation systems.

It turns out that many verification problems of practical importance can be expressed with Boolean equation systems which fall in the stratified class. So a set of applications of our techniques is outlined. For the purposes of this paper, we have also implemented the algorithm and used it to tackle a series of protocol verification problems. The technique turned out to be successful on these initial applications.

The structure of the paper is as follows. In Section 2, we introduce Boolean equation systems and explain the key notions. In Section 3, we discuss state-of-the-art techniques for fixed-point computations and discuss their limitations. In Section 4, we describe how stratified Boolean equation systems can be solved with our techniques, and combine these ideas into a single effective procedure. In Section 5, we prove the NL-hardness of the problem of solving fixed-point equations. In Section 6, we illustrate that many verification problems can be encoded with equation systems in the stratified class. In Section 7 we present the application of our solution techniques to protocol verification problems and results on two sets of benchmarks. Section 8 concludes.

2 Boolean equation systems

We will give in this section a short presentation of Boolean equation systems and define the notions which will be needed.

Suppose we are given a finite set $\mathcal{X} = \{x_1, ..., x_n\}$ of Boolean variables and a linear order on variables in $\mathcal{X}$. Then, Boolean equation systems are defined as follows.

Definition 2.1 A *Boolean equation system* $\mathcal{E}$ is an ordered sequence of labelled fixed-point equations

$$\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1 \leq i \leq n}$$

where $\sigma_i \in \{\nu, \mu\}$, $x_i \in \mathcal{X}$, and $\alpha_i = op_i X$
with $op_i \in \{\bigvee, \bigwedge\}$ and $X \subseteq \mathcal{X}$.

As usual, it is assumed that all left hand side variables of equations are different. The ordering of equations is important, and we keep the order on variables and their indices in synchrony. We suppose that an empty disjunction is written as $false$ and an empty conjunction is written as $true$. We generally use the letter ϵ to stand for the empty Boolean equation system.

We say that a variable x_i *depends on* variable x_j if the right hand side α_i of equation $\sigma_i x_i = \alpha_i$ contains a reference to x_j, or to a variable x_k such that x_k depends on x_j.

The semantics of Boolean equation systems provides a uniquely determined *solution* to each system. Next we will give the characterisation of the semantics. To this end we first need to introduce some notations.

Let $\mathbb{O} = (\{0, 1\}, \bigwedge, \bigvee)$ be a two-point lattice of truth values. The solution will be given relative to an environment $v : \mathcal{X} \to \mathbb{O}$ giving meaning to all variables in a Boolean equation system. Let [] denote the empty environment and $[x_i{:=}a]$ the environment that assigns a to x_i. For two distinct environments v and v' with disjoint variable domains, vv' denotes their union. Thus, for example $[][x_i{:=}a] = [x_i{:=}a] = [x_i{:=}a][]$ and $[x_i{:=}a][x_j{:=}b] = [x_j{:=}b][x_i{:=}a]$. The meaning of a right-hand side α_i is trivially defined:

$$[\![\bigvee\{x_1, x_2, ...x_n\}]\!]v = \bigvee\{v(x_1), v(x_2), ..., v(x_n)\}$$

$$[\![\bigwedge\{x_1, x_2, ...x_n\}]\!]v = \bigwedge\{v(x_1), v(x_2), ..., v(x_n)\}$$

Then, the solution to a Boolean equation system is defined as follows (for similar definitions see e.g. [2,14,21,23,26]).

Definition 2.2 The solution $[\![\mathcal{E}]\!]v$ to a Boolean equation system $\mathcal{E}$ relative to

an environment v is inductively defined by:

$$[\![\epsilon]\!]v = []$$
$$[\![(\sigma_i x_i = \alpha_i)\mathcal{E}]\!]v = ([\![\mathcal{E}]\!]v[x_i{:=}a])[x_i{:=}a]$$

where

$$a = \begin{cases} \mu u.[\![\alpha_i]\!](v[x_i{:=}u]([\![\mathcal{E}]\!]v[x_i{:=}u])) \text{ if } \sigma_i = \mu \\ \nu u.[\![\alpha_i]\!](v[x_i{:=}u]([\![\mathcal{E}]\!]v[x_i{:=}u])) \text{ if } \sigma_i = \nu \end{cases}$$

Notice in the above definition that $\sigma_i u.\alpha_i(u)$ denotes the extremal σ_i fixed point of the function α_i on the lattice $\mathbb{O}$.

3 Solution Techniques and Limitations

In [21,5] there are useful principles which allow to solve Boolean equation systems (see Lemma 6.2 and Lemma 6.3 in [21], or Proposition 1.4.7 in [5]). Since our proofs and techniques are essentially based on these properties we state them here.

Lemma 3.1 *Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be Boolean equation systems, and let $\sigma x = \alpha$ be a fixed-point equation. Let α' be exactly the same expression as α, except that all occurrences of x in α are substituted with $true$ if $\sigma = \nu$, and with $false$ if $\sigma = \mu$. Then $\mathcal{E}_1(\sigma x = \alpha)\mathcal{E}_2$ and $\mathcal{E}_1(\sigma x = \alpha')\mathcal{E}_2$ have the same solutions.*

Lemma 3.2 *Let $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_3$ be Boolean equation systems. Let $\sigma_1 x_1 = \alpha$, $\sigma_1 x_1 = \alpha'$ and $\sigma_2 x_2 = \beta$ be fixed-point equations where α' is exactly the same expression as α except that all occurrences of x_2 are substituted with expression β. Then*

$$\mathcal{E}_1(\sigma_1 x_1 = \alpha)\mathcal{E}_2(\sigma_2 x_2 = \beta)\mathcal{E}_3$$

and

$$\mathcal{E}_1(\sigma_1 x_1 = \alpha')\mathcal{E}_2(\sigma_2 x_2 = \beta)\mathcal{E}_3$$

have the same solutions.

These lemmas allow to solve fixed-point equations with a method very similar to the Gauss elimination in linear algebra. In general, this approach is not well suited to solve large equation systems, because of its exponential time worst case complexity (see e.g. [21]).

The state-of-the-art algorithms for the fixed-point computation are often exponential only in the alternation depth of the underlying system, like those from [1,20].

Also, there are many classes of equation systems for which very efficient polynomial time algorithms exist, see for instance [1,4,6,11,12,14]. Yet a remaining problem is the memory consumption of these algorithms.

Due to the *state explosion problem* in *explicit-state* model checking, even relatively simple verification problems may lead to systems with millions of fixed-point equations and these equations may become of astronomical sizes. Since typical approaches to fixed-point computations rely on the *standard representation* [2] for the equation systems, all equations need to be kept in the memory at the same time. That is, when storing the fixed-point equations explicitly in terms of graphs (like in [1,4,6,11,12,14,20,26]), the memory is often wasted because, upon termination of the fixed-point computations, the memory holds both the graph decoding of the equations and the representation of the solutions for all variables involved.

Instead, we want the solution of a given fixed-point equation system to be determined without resorting to standard representation for the equations, and prefer to find the solutions in a way that would minimize the construction and examination of data structures.

In the next section, we will call the attention to a restricted class of Boolean fixed-point equation systems, which allows for computing their solutions in a memory-efficient way. That is, we identify a class of systems whose solutions can be found without resorting to the standard representation to decode the equation systems.

This is achieved by introducing restrictions on how the variables may be interspersed in the equations. We will then show that knowing beforehand that the system is in such a form allows for using only one bit of storage per equation, which makes the class appealing from a practical point of view.

4 Memory-Efficient Solutions via Stratification

To obtain a fast, memory-efficient solution technique for Boolean equation systems, we introduce the notion of *stratification*. The concept dates back to [3,9] and is well known from the context of *Datalog*. More recently, the concept was successively employed in [24], but the restriction appears to be new in the setting of Boolean equations.

In order to express stratification in Boolean equation systems, we define a function $rank : \mathcal{X} \to \{1, 2, ...\}$ which assigns to each variable a rank number. Based on such a mapping, stratification in equation systems is defined as follows.

Definition 4.1 Let $\mathcal{E}$ be a Boolean equation system $\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1 \leq i \leq n}$. We call system $\mathcal{E}$ *stratified* with respect to a mapping $rank : \mathcal{X} \to \{1, 2, ..., n\}$

iff

(i) the mapping *rank* is bijective, and

(ii) for all $1 \leq i \leq n$, x_i depends only on variables $x_j \in \mathcal{X}$ with $rank(x_j) \leq rank(x_i)$.

A virtue of a Boolean equation system being stratified is that the computation of its solution becomes very simple, and can be optimized to be memory efficient. We present here a simple and effective method to solve such a Boolean equation system. Part of the cash-value of our technique is that it is easily implementable to improve memory consumption without increasing run-time overhead.

Our algorithm is based on the observation that all equations of a stratified Boolean equation system can be solved one at a time, starting from the equations whose left-hand side variables have the lowest ranks. More precisely, our algorithm is based on the following results.

Lemma 4.2 *Let $\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1 \leq i \leq n}$ be a system which is stratified w.r.t. $rank : \mathcal{X} \to \{1, 2, ..., n\}$. Let $\sigma_i x_i = \alpha_i$ be an equation of $\mathcal{E}$ where $op_i = \bigwedge$. Then the following are equivalent:*

(i) *The solution of x_i is* 0

(ii) *$\exists x_j \in \alpha_i$ such that:*
   (a) *$\sigma_i = \mu$ and $x_j = x_i$; or*
   (b) *$rank(x_j) < rank(x_i)$ and the solution of x_j is* 0.

Proof. First we show that (ii) implies (i). If α_i contains a variable x_i with $\sigma_i = \mu$, then using Lemma 3.1, the variable x_i in α_i can be replaced by *false*. Now the right hand side contains only conjunctions and the constant $false$ at least once. Hence, the equation reduces to $\mu x_i = false$ and the solution of x_i must be 0. Similarly, if α_i contains a variable x_j whose solution is known to be 0, then by Lemma 3.1, the variable x_j in α_i can be replaced by *false*, and the solution of x_i must be 0.

Now we show by contraposition that (i) implies (ii). So assume that part (ii) does not hold. In such case Lemma 3.1 cannot justify the solution of x_i to be set as 0. In particular, by Def. 4.1, x_i can only depend on such variables that have lower or equal ranks. Now note that the right hand side α_i cannot contain occurrences of variables whose solution would be 0. Then, using Lemma 3.1, all the variables in α_i can be replaced by *true*. Now the right hand side contains only conjunctions and the constant *true*. Hence, the equation reduces to $\mu x_i = true$ and the solution of x_i must be 1. □

Similar observation holds also for the dual case, and the proof of the fol-

lowing Lemma goes in the same way.

Lemma 4.3 *Let $\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1 \leq i \leq n}$ be a system which is stratified w.r.t. $rank : \mathcal{X} \to \{1, 2, ..., n\}$. Let $\sigma_i x_i = \alpha_i$ be an equation of $\mathcal{E}$ where $op_i = \bigvee$. Then the following are equivalent:*

(i) *The solution of x_i is 1*

(ii) *$\exists x_j \in \alpha_i$ such that:*
   (a) *$\sigma_i = \nu$ and $x_j = x_i$; or*
   (b) *$rank(x_j) < rank(x_i)$ and the solution of x_j is 1.*

Now consider a stratified Boolean fixed-point equation system $\mathcal{E}$. To find the solution we can apply the above Lemmata to all its equations, in the order given by their rank numbers. In this way, we can evaluate the whole system equation by equation, starting with the equation with the lowest rank. We next show how these ideas may be combined into a single efficient procedure.

The Algorithm.

Our algorithm is given in Fig. 1 and explained below. The algorithm receives as its inputs a mapping $rank : \mathcal{X} \to \{1, ..., n\}$ and Boolean equation system which is stratified w.r.t. *rank*. We assume that all variables in $\mathcal{X}$ are represented by their index numbers 1 through n when used as inputs to the algorithm. Starting with the equation that has the lowest rank number, the algorithm evaluates the equations until the equation that has the highest rank is solved, and then terminates.[3]

For all equations, the basic operation in the algorithm involves checking whether it is conjunctive or disjunctive. Based on this, the procedure applies either Lemma 4.2 or Lemma 4.3 to each equation, storing the solutions into a boolean array *value*. The crucial insight underlying this operation occurs in lines $4 - 10$ or $12 - 18$.

We only explain this for conjunctive equations (lines $4-10$), the disjunctive case is dual and goes in the similar way. Given a conjunctive equation $\sigma_i x_i = \alpha_i$, the procedure first sets an initial solution 1 for variable x_i (line 4). Then, the procedure applies Lemma 4.2 to the right-hand side α_i of the equation $\sigma_i x_i = \alpha_i$ (lines $5 - 10$), i.e. it checks whether α_i contains an occurrence of a variable which justifies the solution of x_i to be set as 0. If no such a variable is found, the procedure proceeds to solve the next equation and the solution for x_i is fixed as 1. Otherwise, if α_i contains an occurrence of a variable x_i and $\sigma_i = \mu$, the solution of variable x_i is set to 0 (line 7). Similarly, if right-hand

[3] Notice that the equation with the lowest rank has variable $rank^{-1}(1)$ in its left hand side, and the equation with the highest rank has variable $rank^{-1}(n)$ in its left hand side.

function $solve(\{\sigma_i i = \alpha_i\}_{1\leq i\leq n}, rank : \mathcal{X} \to \{1, ..., n\})$ **is**

```
1   for i := 1 to n do
2     case
3       op_{rank^{-1}(i)} = ∧ :
4         value[rank^{-1}(i)] := 1;
5         for all j ∈ α_{rank^{-1}(i)} do
6           if ((j = rank^{-1}(i)) ∧ σ_{rank^{-1}(i)} = μ) then
7             value[rank^{-1}(i)] := 0;
8           else if (value[j] = 0) then
9             value[rank^{-1}(i)] := 0;
10        od
11      op_{rank^{-1}(i)} = ∨ :
12        value[rank^{-1}(i)] := 0;
13        for all j ∈ α_{rank^{-1}(i)} do
14          if ((j = rank^{-1}(i)) ∧ σ_{rank^{-1}(i)} = ν) then
15            value[rank^{-1}(i)] := 1;
16          else if (value[j] = 1) then
17            value[rank^{-1}(i)] := 1;
18        od
19    esac
20  od
```

Fig. 1. An algorithm for solving a stratified Boolean equation system.

side α_i contains a variable x_j whose solution is already known to be 0, then the solution of variable x_i is set to 0 (line 9).

The correctness of the algorithm relies essentially on Lemma 4.2, Lemma 4.3 and Lemma 3.2, and is easy to verify. This leads to the following.

Theorem 4.4 *The algorithm works correctly on any stratified system of Boolean equations, and the array value forms the global solution of the given system when the algorithm terminates.*

In order to formally estimate computational costs, let the *size* of a Boolean equation system $\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1\leq i\leq n}$ be defined as

$$|\mathcal{E}| = \sum_{i=1}^{n} 1 + |\alpha_i|$$

where $|\alpha_i|$ is the number of variables occurring in α_i. Then the next theorem characterizes the complexity of our algorithm.

Theorem 4.5 *A Boolean equation system*

$$\mathcal{E} = \{\sigma_i x_i = \alpha_i\}_{1 \leq i \leq n}$$

which is stratified w.r.t. a mapping rank : $\mathcal{X} \to \{1, ..., n\}$ *can be solved in time* $O(|\mathcal{E}|)$ *and using only space* $O(n)$.

Admittedly, in order to obtain the above time and space bounds one needs a sequential random access to the right-hand sides of equations. Usually, this is easily achieved without any additional computational overhead, because in typical applications (as those in Section 6) the equations are trivially presorted according to a suitable rank function.

We conclude this section by comparing our space efficiency with conventional techniques. Like Theorem 4.5 shows, our algorithm achieves a lower space complexity than existing Boolean equation system solution algorithms [1,2,14,21,26,18,20] executed on stratified systems. These algorithms have the worst-case space complexity $O(|\mathcal{E}|)$. The experimental results in Section 7 indicate that our procedure works better than [14] in practice.

The first memory efficient resolution algorithm for acyclic Boolean equation systems was presented in [22], which obtains the space complexity $O(n)$ comparable to our result on stratified Boolean equation systems. The basic difference between the two approaches lies in that the algorithm in [22] is *local* whereas our algorithm is *global*.

Of course, it must be noticed that more generic algorithms for non-stratified Boolean equation systems might often be optimized as well, by taking into account the dependencies between variables to avoid useless re-computations of known-stabilized variables, in the same way as suggested by algorithm in Fig. 1.

5 NL-Hardness Result

In this section, we investigate the space complexity of solving general Boolean equation systems. It is widely believed that the problem is in the complexity class P, but this containment is currently in doubt. It is also well known that $L \subseteq NL \subseteq P$.[4] So an interesting question is to ask what are the space bounds with respect to the logarithmic space classes. In particular, we can show that the problem of solving Boolean equation systems is NL-hard. This result appears to be new.

[4] Recall that L denotes the class of problems decidable by deterministic Turing machines in logarithmic space, and NL denotes the corresponding class of problems decidable by nondeterministic Turing machines [25].

We prove the NL-hardness result by showing that the graph reachability problem is log-space reducible to the problem of solving Boolean equation systems. So let $G = (V, E)$ denote a directed graph with a node set $V = \{1, 2, ..., n\}$ for some $n \geq 1$ and edge set $E \subseteq V \times V$. The graph reachability problem, is the following decision problem.

INPUT: a directed graph $G = (\{1, 2, ..., n\}, E)$

QUESTION: decide whether G has a path from node 1 to node n

The problem is known to be NL-complete, and previously variety of other problems were shown to be complete for NL in [15,25], via reduction from reachability. This leads to a statement of our space complexity bound.

Theorem 5.1 *The problem of solving Boolean equation systems is NL-hard.*

Proof. The proof is based on a mapping from the graph G to a sequence of Boolean fixed-point equalities. Suppose that $G = (\{1, 2, ..., n\}, E)$ is a directed graph for some $n > 1$. We construct a sequence of equations:

$$\mu x_1 = \bigvee \{x_i : (1, i) \in E\}$$
$$\mu x_2 = \bigvee \{x_i : (2, i) \in E\}$$
$$\vdots$$
$$\mu x_{n-1} = \bigvee \{x_i : (n-1, i) \in E\}$$
$$\nu x_n = \bigvee \{x_n\}$$

It is then straightforward to verify that the solution of the resulting instance of Boolean equation system is such that x_1 has value 1 iff there is a path from node 1 to node n in G. Moreover, the construction from G to fixed-point equations is clearly a logarithmic space reduction. □

Notice that the Boolean equation system constructed in the above proof is disjunctive, and that there are no mutually dependent variables with different signs, i.e. the resulting system is alternation-free. Thus, the problem of solving Boolean equation systems remains NL-hard even for disjunctive, alternation-free systems.

6 Applications

The fixpoint analysis techniques can be used to tackle a variety of verification problems. The overall idea of the approach can be summarized as follows. The general procedure consists of first generating the fixed-point equalities from the verification problem, and then solving the equation systems using a

suitable solver. These two phases are clearly independent of each other, but knowing beforehand that a verification problem leads to stratified equation systems allows for tuning the solver, for example like discussed in Section 4.

Now we give some examples of verification problems that can be encoded as stratified Boolean equation systems, and therefore can be solved efficiently with our techniques.

6.1 Model Checking

The model checking problem of modal μ-calculus on *acyclic* labelled transition systems reduces to the task of computing the solutions of stratified Boolean equation systems. Various important application areas are discussed in [22] which include, for instance, *run-time monitoring* and *trace analysis*.

Using the standard translation, all μ-calculus formulas applied to any acyclic transition system lead to stratified Boolean equation systems.

Theorem 6.1 *The representation of μ-calculus model checking problem as Boolean equation systems is stratified, if the labelled transition system is acyclic and the encoding in [5,21] is used.*

As soon as we realize this, we obtain a model checking technique for full μ-calculus logic on acyclic models, which has a slightly better estimation on its worst-case time complexity than the techniques from [22]. If applied to the model checking of μ-calculus on *acyclic* models, our approach avoids an unpleasant quadratic blow up in the size of formulas.

However, one of the merits of the algorithm in [22] is that it works in a demand-driven way, which does not seem to be easily achieved with our approach. This is not surprising taking into consideration the fact that our resolution algorithm is *global*, as opposed to the *local* approach from [22].

In addition, for the syntactic fragment of μ-calculus which does not contain fixpoint operators, we have the following:

Theorem 6.2 *The representation of Hennessy-Milner logic model checking problem as Boolean equation systems is stratified, if the encoding in [5,23] is used.*

6.2 Equivalence/Preorder Checking

Beside their utility in model checking, stratified Boolean equation systems arise also in the context of equivalence and preorder checking. For instance, given two labelled transition systems, whenever at least one of them is acyclic, the Boolean equation systems related to *strong bisimulation* and its preorder

become stratified. Indeed, like observed in [23], the encoding of strong bisimulation on an acyclic labelled transition system yields a Boolean equation system without mutually dependent variables. This leads to the following result.

Theorem 6.3 *The representations of simulation and bisimulation equivalences as Boolean equation systems are stratified, if the labelled transition system is acyclic and the encoding in [18] is used.*

7 Experimental Results

Perhaps, the best way to compare the efficiency of implementations of Boolean equation system solvers is by applying them to a set of benchmarks. In this section, we provide initial experimental results on equation system benchmarks from realistic applications. To assess the practical performance of our algorithm we have implemented it in the C programming language.

All testing was done on a 1.0Ghz AMD Athlon running Linux with sufficient main memory. The times reported are the average of 3 runs of the times for the solvers to find solutions as reported by the */usr/bin/time* command.

As benchmarks we used protocol models taken from [8], instantiated with the μCRL-toolset [7], and converted to Boolean equation systems. The first verification problem consists of model checking a μ-calculus formula on a sequence of distributed leader election protocol models of increasing size, as described below. The second series of experiments deals with the problem of checking simulation equivalence between the leader election protocol and its abstraction.

7.1 Model Checking DKR Leader Election Protocol

In brief, the Dolev-Klawe-Rodeh (DKR) leader election protocol [10] consists of n parties connected in a ring. These parties exchange messages and after a finite number of messages, the party with the highest identification informs the others being a leader. The protocol is modelled in [13] and the desired safety property we need to check is that "Two *leader*-actions can never occur on a same execution path of the system". This specification is expressed with a μ-calculus formula (1) below

$$\nu X.([true]X \wedge [leader]\phi) \tag{1}$$

where the subformula ϕ is (2):

$$\nu Y.([true]Y \wedge [leader]false) \tag{2}$$

The formula (1) is globally true on all protocol models.

Benchmark	Tool	Time(sec)	Mem(B)
DKR(5)	cbess	0.01	42 232
$\|\mathcal{E}\|$:	sbess	0.01	6 932
7 192	(%)	(0)	(16, 4)
DKR(6)	cbess	0.05	210 088
$\|\mathcal{E}\|$:	sbess	0.04	31 636
36 714	(%)	(**80.0**)	(15, 1)
DKR(7)	cbess	0.24	1 054 536
$\|\mathcal{E}\|$:	sbess	0.21	146 052
190 618	(%)	(87, 5)	(13, 8)
DKR(8)	cbess	0.80	2 115 304
$\|\mathcal{E}\|$:	sbess	0.70	425 388
632 284	(%)	(87, 5)	(20, 1)
DKR(9)	cbess	4.17	16 537 664
$\|\mathcal{E}\|$:	sbess	3.64	1 943 428
3 162 712	(%)	(87, 3)	(**11,8**)

Fig. 2. Total solution times and memory consumption for Boolean equation system solvers on model checking benchmark data.

The above verification problem can be directly formalized as a Boolean equation system, prompting us to encode it as follows:

$$\left.\begin{aligned} \nu\, x_s &= \bigwedge_{s' \in \nabla(t,s)} x_{s'} \wedge \bigwedge_{s' \in \nabla(l,s)} y_{s'} \\ \nu\, y_s &= \bigwedge_{s' \in \nabla(l,s)} false \wedge \bigwedge_{s' \in \nabla(t,s)} y_{s'} \end{aligned}\right\} \forall s \in S$$

Here, $\nabla(l,s) := \{s' | s \overset{leader}{\longrightarrow} s'\}$ and $\nabla(t,s) := \{s' | s \overset{i}{\longrightarrow} s' \text{ and } i \in A\}$.

At first sight, these equations do not seem to be a stratified Boolean equation system, but as the concrete labelled transition systems instantiated from high level specifications in [13] are acyclic, all equations will become stratified.

We then evaluated the performance of our algorithm by measuring solution times and comparing memory usage to another Boolean equation system solver

from [14]. Although the algorithm from [14] (see Fig. 1 in [14]) is designed to solve systems which may contain alternating fixed-points, it is guaranteed to exhibit linear time performance for stratified Boolean systems. The essential difference between the two algorithms is that the one from [14] relies on the standard representation [2] for decoding the equation systems, which allows for a fair comparison.

The results are given in Fig. 2 where the columns are:

- Benchmark: $DKR(n)$ DKR leader election protocol model with n parties, and $|\mathcal{E}|$ the size of the boolean equation system corresponding to the verification problem.
- Tool: *cbess* a solver for conjunctive boolean equation systems from [14]; *sbess* a stratified boolean equation system solver in Fig. 1; the row (%) shows the obtained performance improvement; the number in the parentheses indicates the performance measure of the new algorithm as a percentage of the performance of the algorithm in [14].
- Time(sec): The time in seconds needed to solve the equation system.
- Mem(B): The number of memory bytes needed to solve the equation system.

Based on the experimental results one may draw the following conclusions. Our algorithm outperforms the one from [14] in time. The degrees of improvement range up to a moderate speedup of around 20%. In addition, we obtain considerable reductions of memory consumption whenever our algorithm is used instead of the one from [14]. Compared to the technique in [14], using the solver in Fig. 1 always increases memory savings, going up to around 88%. This result is due to the fact that our algorithm exploits the particular stratified structure of Boolean equation systems to avoid storing all the equations. Instead of representing equation systems with *standard representation* [2], it suffices to hold in the memory only one bit per the solution of each variable involved.

7.2 Simulation Checking DKR Leader Election Protocol

The second application we consider deals with the problem of simulation checking the DKR leader election protocol. The second experiments compare each leader election protocol LTS modulo simulation relation with a more abstract LTS, and verify that the reduced LTS is a correct abstraction of the concrete protocol LTSs.

We used the encoding in [18] to represent the simulation equivalence between the concrete and abstract LTSs. Consequently, all equations of the corresponding Boolean equation systems will be stratified, because the LTSs

Benchmark	Tool	Time(sec)	Mem(B)
DKR(5)	cbess	0.01	24 580
$\|\mathcal{E}\|$:	sbess	0.00	14 232
3 553	(%)	(0)	(**57,9**)
DKR(6)	cbess	0.03	120 860
$\|\mathcal{E}\|$:	sbess	0.01	73 456
18 359	(%)	(**33,3**)	(60, 8)
DKR(7)	cbess	0.11	600 292
$\|\mathcal{E}\|$:	sbess	0.09	381 264
95 311	(%)	(81, 8)	(63, 5)
DKR(8)	cbess	0.35	1 902 628
$\|\mathcal{E}\|$:	sbess	0.30	1 264 596
316 144	(%)	(85, 7)	(66, 5)
DKR(9)	cbess	1.70	9 220 524
$\|\mathcal{E}\|$:	sbess	1.49	6 325 444
1 581 356	(%)	(87, 6)	(68, 6)

Fig. 3. Total solution times and memory consumption for Boolean equation system solvers on simulation checking benchmark data.

instantiated from high level specifications in [13] are acyclic.

For each experiment, the table in Fig. 3 shows the measures obtained using the algorithm in Fig. 1 and [14], and the corresponding difference ratios (the columns are explained as in Fig. 2).

A conclusion to be drawn from the second example is that the memory usage is substantially improved in all of the experiments; the reduction is down to around 58%.

8 Conclusion

In this paper we have presented a method for computing the solutions to restricted, stratified Boolean equation systems. Our method consists of a fast, memory-efficient procedure which needs only a single pass through any

given system of stratified Boolean equations. We proved the correctness and complexity of our technique.

Furthermore, we have studied the space complexity of solving Boolean equation systems. We showed that the space complexity of solving Boolean equation systems remains NL-hard even for a relatively simple class of fixpoint equations.

Also, we have demonstrated that stratified Boolean equation systems provide a suitable framework for expressing many verification problems on finite-state concurrent systems, for instance model checking of μ-calculus as well as equivalence/preorder checking on acyclic models.

We have demonstrated the performance of our new algorithm using leader election protocol verification examples. The implementation of our technique has been proven both efficient and scalable. The benchmarks show significant memory savings and small reductions in solution times.

References

[1] H.R. Andersen. Model checking and Boolean graphs. *Theoretical Computer Science*, 126:3–30, 1994.

[2] H.R. Andersen, B. Vergauwen. Efficient Checking of Behavioural Relations and Modal Assertions using Fixed-Point Inversion. In *Proceedings of Conf. on Computer Aided Verification*, Lecture Notes on Computer Science 939, pages 142–154, Springer Verlag, 1995.

[3] K. Apt, H. Blair and A. Walker. A theory of declarative programming. *Foundations of deductive databases and logic programming*, pages 89–149, Morgan Kaufman, 1988.

[4] A. Arnold and P. Crubille. A linear time algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29: 57–66, 1988.

[5] A. Arnold and D. Niwinski. Rudiments of μ-calculus. *Studies in Logic and the foundations of mathematics*, Volume 146, Elsevier, 2001.

[6] G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ-calculus. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1055, pages 107–126, Springer Verlag, 1996.

[7] S. Blom, W. Fokkink, J.F. Groote, I. van Langevelde, B. Lisser and J. van de Pol. μCRL: a toolset for analysing algebraic specifications. In *Proceedings of Conf. on Computer Aided Verification*, Lecture Notes in Computer Science 2102, pages 250–254, Springer Verlag, 2001.

[8] S. Blom and J. van de Pol. State space reduction by proving confluence. In *Proceedings of Conf. on Computer Aided Verification*, Lecture Notes on Computer Science 2404, pages 596–609, Springer Verlag, 2002.

[9] A. Chandra and D. Harel. Computable queries for relational databases. *Journal of Computer and System Sciences*, 21(2): 156–178, 1980.

[10] D. Dolew, M. Klawe, and M. Rodeh. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *Journal of Algorithms*, 3(3): 245–260, 1982.

[11] E.A. Emerson, C. Jutla and A.P. Sistla. On model checking for fragments of the μ-calculus. In *Proceedings of the Fifth International Conference on Computer Aided Verification*, Lecture Notes in Computer Science 697, pages 385–396, Springer Verlag, 1993.

[12] E.A. Emerson, C. Jutla, and A.P. Sistla. On model checking for the μ-calculus and its fragments. *Theoretical Computer Science*, 258:491–522, 2001.

[13] L. Fredlund, J.F. Groote and H. Korver. Formal Verification of a Leader Election Protocol in Process Algebra. *Theoretical Computer Science*, 177: 459–486, 1997.

[14] J.F. Groote and M. Keinänen. Solving Disjunctive/Conjunctive Boolean Equation Systems with Alternating Fixed Points. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 2988, pages 436 – 450, Springer Verlag, 2004.

[15] N. Jones, Y. Lien, and W. Laaser. New Problems Complete for Nondeterministic Log Space. *Mathematical Systems Theory*, 10: 1–17, 1976.

[16] M. Jurdzinski. Deciding the winner in parity games is in $UP \cap co-UP$. *Information Processing Letters*, 68:119–124, 1998.

[17] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[18] K. Larsen. Efficient Local Correctness Checking. In *Proceedings of Conf. on Computer Aided Verification*, Lecture Notes on Computer Science 663, pages 30–43, Springer Verlag, 1992.

[19] X. Liu and S.A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points. In *Proceedings of the 26th International Conference on Automata, Languages, and Programming*, Lecture Notes in Computer Science 1443, pages 53–66, Springer Verlag, 1998.

[20] X. Liu, C.R. Ramakrishnan and S.A. Smolka. Fully Local and Efficient Evaluation of Alternating Fixed Points. In *Proceedings of the 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1384, pages 5–19, Springer Verlag, 1998.

[21] A. Mader. Verification of Modal Properties using Boolean Equation Systems. PhD thesis, Technical University of Munich, 1997.

[22] R. Mateescu. Local Model-Checking of Modal Mu-Calculus on Acyclic Labeled Transition Systems. In *Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 2280, pages 281–295, Springer Verlag, 2002.

[23] R. Mateescu. A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 2619, pages 81–96, Springer Verlag, 2003.

[24] F. Nielson, H. Nielson, H. Sun, M. Buchholtz, R. Hansen, H. Pilegaard and H. Seidl. The Succinct Solver Suite. In *Proceedings of the 10th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 2988, pages 251–265, Springer Verlag, 2004.

[25] C. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.

[26] B. Vergauwen and J. Lewi. Efficient local correctness checking for single and alternating Boolean equation systems. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 820, pages 302–315, Springer Verlag, 1994.